Team Wind: Group research project for developing and experimenting biomedical images analysis for Covid19

Created by: Leiyi Gao, Steven Liao, Miho Shimizu, Alexander Shirazi, Tyler Heslop, Sungho Lee.

Present by: Leiyi Gao & Tyler Heslop

Summary/Outline

- Develop a 3D diagnosis system for Covid19(lung) biomedical images.
- Our system includes:
- Segmentation:
  - Using the process to detect boundaries of anatomical structures of 3D covid19 lung images.
  - Divide the infected region into a group and showing the result.
- Registration:
  - Using two sets of covid19 images(Infected with different stage and healthy lung) to align the images for analysis.
  - Visualized the difference between covid19 infected lung and the healthy lung.
- Quantification:
  - Experimenting different number/inputs inside of code to find the best value.
  - Validating our Registration result visually/quantitatively.
  - Compare different segmentation/registrations algorithms to determine which one perform best for the used data.

Algorithms we implemented:

- Segmentation:
  - Region Growing Algorithm-Pixel based image segmentation method since it involves the selection of initial seed. It examines neighboring pixels of initial seed points and determines whether the pixel neighbors should be added to the region.
  - Watershed Algorithm-It is a transformation defined on a grayscale image and separates different objects in an image. This allows for counting the objects or for further analysis of the separated objects.
- Registration:
  - Demon Algorithm-It is used for non-rigid registration of 3D medical images. This algorithm is based on intuitive ideas about image registration, it could be slow and not accurate.
  - Intensity Based Algorithm-It is an iterative process, and it requires that user specify a pair of images, a metric, an optimizer, and a transformation type. The metric defines the image similarity metric for evaluating the accuracy of the registration.

Segmentation
Region Growing Algorithm:

- The main goal is to partition an image into regions. Some segmentation methods such as thresholding achieve this goal by looking for the boundaries between regions based on discontinuities in grayscale or color properties. Region-based segmentation is a technique for determining the region directly. The basic formulation is:

$$\bigcup_{i=1}^{n} R_i = R.$$

$$R_i \text{ is a connected region, } \mathrm{i} = 1,\ 2,\ \ldots, \mathrm{n}$$

$$R_i \bigcap R_j = \varnothing, i \neq j$$

$$P(R_i) = \text{TRUE for } i = 1, 2, \ldots, n.$$

$$P(R_i \bigcup R_j) = \text{FALSE for any adjacent region } R_i \text{ and } R_j.$$

Region Growing Algorithm Continued

- Read the data and select the seed point:

```
img_T1 = sitk.ReadImage("./covid19.mha")
# [0,255],the default values for the output
img_T1_255 = sitk.Cast(
    sitk.IntensityWindowing(
        img_T1,
        windowMinimum=T1_WINDOW_LEVEL[1] - T1_WINDOW_LEVEL[0] / 2.0,
        windowMaximum=T1_WINDOW_LEVEL[1] + T1_WINDOW_LEVEL[0] / 2.0,
    ),
    sitk.sitkUInt8,
)
```

```
point_acquisition_interface = gui.PointDataAquisition(img_T1, window_level=(500,50))
idx = img_T1.GetSize()

# preselected seed point in the left ventricle
point_acquisition_interface.set_point_indexes([(int(idx[0]/2), int(idx[1]/2), int(idx[2]-5))])
```

Output:

Region Growing Algorithm Continued

- Connected Threshold: A simple criterion for including pixels in a growing region is to evaluate intensity value inside a specific interval. Lower and upper threshold values should be provided. The region-growing algorithm includes those pixels whose intensities are inside the interval.

$$I(\mathbf{X}) \in [\text{lower}, \text{upper}]$$

- We now set the lower and upper threshold values. Any pixel whose value is between lower-Threshold and upper-Threshold will be included in the region, and any pixel whose value is outside will be excluded. Setting these values too close together will be too restrictive for the region to grow; setting them too far apart will cause the region to engulf the image.

Region Growing Algorithm Continued

- Connected Threshold:

```
seg_explicit_thresholds = sitk.ConnectedThreshold(
    img_T1, seedList=initial_seed_point_indexes, lower=150, upper=180
)

gui.MultiImageDisplay(
    image_list=[sitk.LabelOverlay(img_T1_255, seg_explicit_thresholds)],
    title_list=["connected threshold result"],
)
```

- Output:

Region Growing Algorithm Continued

- **Confidence Connected:** This filter extracts a connected set of pixels whose pixel intensities are consistent with the pixel statistics of a seed point. The mean and variance across a neighborhood (8-connected, 26-connected, etc.) are calculated for a seed point. Then pixels connected to this seed point whose values are within the confidence interval for the seed point are grouped.

- The width of the confidence interval is controlled by the "Multiplier" variable (the confidence interval is the mean plus or minus the "Multiplier" times the standard deviation). If the intensity variations across a segment were gaussian, a "Multiplier" setting of 2.5 would define a confidence interval wide enough to capture 99% of samples in the segment.

Region Growing Algorithm Continued

- Connected Threshold:

```
seg_implicit_thresholds = sitk.ConfidenceConnected(
    img_T1,
    seedList=initial_seed_point_indexes,
    numberOfIterations=0,
    multiplier=2.5,
    initialNeighborhoodRadius=1,
    replaceValue=1,
)

gui.MultiImageDisplay(
    image_list=[sitk.LabelOverlay(img_T1_255, seg_implicit_thresholds)],
    title_list=["confidence connected result"],
)
```

- Output:

Region Growing Algorithm Continued

- **Vector Confidence Connected:** This filter extracts a connected set of pixels whose pixel intensities are consistent with the pixel statistics of a seed point. The mean and variance across a neighborhood (8-connected, 26-connected, etc.) are calculated for a seed point. Then pixels connected to this seed point whose values are within the confidence interval for the seed point are grouped. The width of the confidence interval is controlled by the "Multiplier" variable (the confidence interval is the mean plus or minus the "Multiplier" times the standard deviation). If the intensity variations across a segment were gaussian, a "Multiplier" setting of 2.5 would define a confidence interval wide enough to capture 99% of samples in the segment.

- After this initial segmentation is calculated, the mean and variance are re-calculated. All the pixels in the previous segmentation are used to calculate the mean the standard deviation (as opposed to using the pixels in the neighborhood of the seed point). The segmentation is then recalculated using these refined estimates for the mean and variance of the pixel values. This process is repeated for the specified number of iterations. Setting the "NumberOfIterations" to zero stops the algorithm after the initial segmentation from the seed point.

Region Growing Algorithm Continued

- **Vector Confidence Connected:**

```
img_T2 = sitk.ReadImage("./covid19.mha")
img_multi = sitk.Compose(img_T1, img_T2)
```

```
seg_implicit_threshold_vector = sitk.VectorConfidenceConnected(
    img_multi, initial_seed_point_indexes, numberOfIterations=2, multiplier=2.5
)

gui.MultiImageDisplay(
    image_list=[sitk.LabelOverlay(img_T1_255, seg_implicit_threshold_vector)],
    title_list=["vector confidence connected result"],
)
```

- **Output:**

Segmentation
Watershed Algorithm:

- Watershed algorithm is based on extracting sure background and foreground and then using markers will make watershed run and detect the exact boundaries. This algorithm generally helps in detecting touching and overlapping objects in image.

- During the successive flooding of the grey value relief, watersheds with adjacent catchment basins are constructed. This flooding process is performed on the gradient image, the basins should emerge along the edges.

Watershed Algorithm Continued

- Original file :

Read data:

```
%matplotlib inline
import matplotlib.pyplot as plt
import SimpleITK as sitk
from myshow import myshow, myshow3d
%run update_path_to_download_script
from downloaddata import fetch_data as fdata
```

```
img = sitk.ReadImage("./DICOM3.mha")
myshow(img)
```

Output:

Gradient Watersheds Segmentation:

```
sigma = img.GetSpacing()[0]
level = 4
```

```
feature_img = sitk.GradientMagnitude(img)
myshow(feature_img, "Edge Features")
```

Output:

Watershed Algorithm Continued:

- **Gradient Watersheds Segmentation:**

```
ws_img = sitk.MorphologicalWatershed(
    feature_img, level=0, markWatershedLine=True, fullyConnected=False
)
myshow(sitk.LabelToRGB(ws_img), "Watershed Over Segmentation")
from ipywidgets import interact, interactive, FloatSlider


def callback(feature_img, *args, **kwargs):
    ws_img = sitk.MorphologicalWatershed(feature_img, *args, **kwargs)

    myshow(sitk.LabelToRGB(ws_img), "Watershed Segmentation")


interact(
    lambda **kwargs: callback(feature_img, **kwargs),
    markWatershedLine=True,
    fullyConnected=False,
    level=FloatSlider(min=0, max=255, step=0.1, value=4.0),
)
```

- **Output:**

Registration
Intensity Based:

- It is an iterative process, and it requires that user specify a pair of images, a metric, an optimizer, and a transformation type. The metric defines the image similarity metric for evaluating the accuracy of the registration.
- Steps:

Intensity Based Continued

- Usually, there are 2 sublots created. Moving and Fixed.

```
def display_images(fixed_image_z, moving_image_z, fixed_npa, moving_npa):
    # Create a figure with two subplots and the specified size.
    plt.subplots(1, 2, figsize=(10, 8))

    # Draw the fixed image in the first subplot.
    plt.subplot(1, 2, 1)
    plt.imshow(fixed_npa[fixed_image_z, :, :], cmap=plt.cm.Greys_r)
    plt.title("fixed image")
    plt.axis("off")

    # Draw the moving image in the second subplot.
    plt.subplot(1, 2, 2)
    plt.imshow(moving_npa[moving_image_z, :, :], cmap=plt.cm.Greys_r)
    plt.title("moving image")
    plt.axis("off")

    plt.show()
```

- Output:

Intensity Based Continued

- Then we lay them together(Euler method in our lecture):

```
initial_transform = sitk.CenteredTransformInitializer(
    fixed_image,
    moving_image,
#Euler3DTransform
    sitk.AffineTransform(3),
    sitk.CenteredTransformInitializerFilter.GEOMETRY,
)

moving_resampled = sitk.Resample(
    moving_image,
    fixed_image,
    initial_transform,
    sitk.sitkLinear,
    0.0,
    moving_image.GetPixelID(),
)

interact(
    display_images_with_alpha,
    image_z=(0, fixed_image.GetSize()[2] - 1),
    alpha=(0.0, 1, 0.05),
    fixed=fixed(fixed_image),
    moving=fixed(moving_resampled),
);
```

- Output:

Intensity Based Continued:

- We will need our metric value and optimizer's stopping point:

```
registration_method = sitk.ImageRegistrationMethod()

# Similarity metric settings.
registration_method.SetMetricAsMattesMutualInformation(numberOfHistogramBins=50)
registration_method.SetMetricSamplingStrategy(registration_method.RANDOM)
registration_method.SetMetricSamplingPercentage(0.01)

registration_method.SetInterpolator(sitk.sitkLinear)

# Optimizer settings.
registration_method.SetOptimizerAsGradientDescent(
    learningRate=1.0,
    numberOfIterations=100,
    convergenceMinimumValue=1e-6,
    convergenceWindowSize=10,
)
registration_method.SetOptimizerScalesFromPhysicalShift()

# Setup for the multi-resolution framework.
registration_method.SetShrinkFactorsPerLevel(shrinkFactors=[4, 2, 1])
registration_method.SetSmoothingSigmasPerLevel(smoothingSigmas=[2, 1, 0])
registration_method.SmoothingSigmasAreSpecifiedInPhysicalUnitsOn()

# Don't optimize in-place, we would possibly like to run this cell multiple times.
registration_method.SetInitialTransform(initial_transform, inPlace=False)

# Connect all of the observers so that we can perform plotting during registration.
registration_method.AddCommand(sitk.sitkStartEvent, start_plot)
registration_method.AddCommand(sitk.sitkEndEvent, end_plot)
registration_method.AddCommand(
    sitk.sitkMultiResolutionIterationEvent, update_multires_iterations
)
registration_method.AddCommand(
    sitk.sitkIterationEvent, lambda: plot_values(registration_method)
)

final_transform = registration_method.Execute(
    sitk.Cast(fixed_image, sitk.sitkFloat32), sitk.Cast(moving_image, sitk.sitkFloat32)
)
```

Intensity Based Continued:

- Let's see our result:

An optimizer is required to explore the parameter space of the transform in search of optimal values of the metric.

The metric will compare how well the two images match each other. Metric types are usually templated over the image types.

```
print(f"Final metric value: {registration_method.GetMetricValue()}")
print(f"Optimizer's stopping condition, {registration_method.GetOptimizerStopConditionDescription()}")
```

```
Final metric value: -0.49282169894574374
Optimizer's stopping condition, GradientDescentOptimizerv4Template: Convergence checker passed at iteration 52.
```

Intensity Based Continued:

- Then we can get our final output:

```
moving_resampled = sitk.Resample(
    moving_image,
    fixed_image,
    final_transform,
    sitk.sitkLinear,
    0.0,
    moving_image.GetPixelID(),
)

interact(
    display_images_with_alpha,
    image_z=(0, fixed_image.GetSize()[2] - 1),
    alpha=(0.0, 1.0, 0.05),
    fixed=fixed(fixed_image),
    moving=fixed(moving_resampled),
);
```

Registration Demon:

- It is widely used for deformable registration of images from the same imaging modality by iteratively solving the gradient constraint equation to build pixel correspondence between the reference and float images.

- The algorithm itself does not solve the equation completely and it assumes that pixels move in the direction of the gradient, which is not true for most circumstances.

Demon Continued

Same as Intensity Based, we need fixed and moving images
Metric value and Iteration number.

```
def display_images(fixed_image_z, moving_image_z, fixed_npa, moving_npa):
    # Create a figure with two subplots and the specified size.
    plt.subplots(1, 2, figsize=(10, 8))

    # Draw the fixed image in the first subplot.
    plt.subplot(1, 2, 1)
    plt.imshow(fixed_npa[fixed_image_z, :, :], cmap=plt.cm.Greys_r)
    plt.title("fixed image")
    plt.axis("off")

    # Draw the moving image in the second subplot.
    plt.subplot(1, 2, 2)
    plt.imshow(moving_npa[moving_image_z, :, :], cmap=plt.cm.Greys_r)
    plt.title("moving image")
    plt.axis("off")

    plt.show()
fixed_image = sitk.ReadImage("./DICOM3.mha", sitk.sitkFloat32)
moving_image = sitk.ReadImage("./healthylung.mha", sitk.sitkFloat32)


interact(
    display_images,
    fixed_image_z=(0, fixed_image.GetSize()[2] - 1),
    moving_image_z=(0, moving_image.GetSize()[2] - 1),
    fixed_npa=fixed(sitk.GetArrayViewFromImage(fixed_image)),
    moving_npa=fixed(sitk.GetArrayViewFromImage(moving_image)),
);
```

Demon Algorithm Continued

- As we noticed, we used covid19-positive lung and healthy lung

Output:

Demon Algorithm Continued

- We also use Euler method

```
initial_transform = sitk.CenteredTransformInitializer(
    fixed_image,
    moving_image,
#Euler3DTransform
    sitk.AffineTransform(3),
    sitk.CenteredTransformInitializerFilter.GEOMETRY,
)

moving_resampled = sitk.Resample(
    moving_image,
    fixed_image,
    initial_transform,
    sitk.sitkLinear,
    0.0,
    moving_image.GetPixelID(),
)

interact(
    display_images_with_alpha,
    image_z=(0, fixed_image.GetSize()[2] - 1),
    alpha=(0.0, 1.0, 0.05),
    fixed=fixed(fixed_image),
    moving=fixed(moving_resampled),
);
```

- Output:

Demon Algorithm Continued

- The actual Demon Method:

```
def demons_registration(
    fixed_image, moving_image, fixed_points=None, moving_points=None
):

    registration_method = sitk.ImageRegistrationMethod()

    # Create initial identity transformation.
    transform_to_displacment_field_filter = sitk.TransformToDisplacementFieldFilter()
    transform_to_displacment_field_filter.SetReferenceImage(fixed_image)
    # The image returned from the initial_transform_filter is transferred to the transform and cleared out.
    initial_transform = sitk.DisplacementFieldTransform(
        transform_to_displacment_field_filter.Execute(sitk.Transform())
    )

    # Regularization (update field - viscous, total field - elastic).
    initial_transform.SetSmoothingGaussianOnUpdate(
        varianceForUpdateField=0.0, varianceForTotalField=2.0
    )

    registration_method.SetInitialTransform(initial_transform)

    registration_method.SetMetricAsDemons(
        10
    )  # intensities are equal if the difference is less than 10HU

    # Multi-resolution framework.
    registration_method.SetShrinkFactorsPerLevel(shrinkFactors=[4, 2, 1])
    registration_method.SetSmoothingSigmasPerLevel(smoothingSigmas=[8, 4, 0])

    registration_method.SetInterpolator(sitk.sitkLinear)

        learningRate=1.0,
        numberOfIterations=50,
        convergenceMinimumValue=1e-6,
        convergenceWindowSize=10,
    )
    registration_method.SetOptimizerScalesFromPhysicalShift()
    registration_method.AddCommand(sitk.sitkStartEvent, start_plot)
    registration_method.AddCommand(sitk.sitkEndEvent, end_plot)
    registration_method.AddCommand(
        sitk.sitkMultiResolutionIterationEvent, update_multires_iterations
```

Demon Algorithm Continued:

- The metric value and the output:

```
Final metric value: 211830.4788376108
Optimizer's stopping condition, GradientDescentOptimizerv4Template: Convergence checker passed at iteration 9.
```

Quantification

- Region Growing Segmentation

Here we show the difference with two seed index value for Connected Threshold Filter:

```
img_T1, seedList=initial_seed_point_indexes, lower=10, upper=10000
```

connected threshold result

```
img_T1, seedList=initial_seed_point_indexes, lower=150, upper=200
```

connected threshold result

Quantification Continued

- Region Growing Segmentation

Here we show the difference for two Multiplier index value of Confidence-Connected Filter:

```
seg_implicit_thresholds = sitk.ConfidenceConnected(
    img_T1,
    seedList=initial_seed_point_indexes,
    numberOfIterations=0,
    multiplier=6,
    initialNeighborhoodRadius=1,
    replaceValue=1,
)
```

```
seg_implicit_thresholds = sitk.ConfidenceConnected(
    img_T1,
    seedList=initial_seed_point_indexes,
    numberOfIterations=0,
    multiplier=2.5,
    initialNeighborhoodRadius=1,
    replaceValue=1,
)
```

confidence connected result

confidence connected result

Quantification Continued

- Region Growing Clean Up by implementing Closing process(remove small connected parts)

```
vectorRadius = (1, 1, 1)
kernel = sitk.sitkBall
seg_implicit_thresholds_clean = sitk.BinaryMorphologicalClosing(
    seg_implicit_thresholds, vectorRadius, kernel
)
```

```
gui.MultiImageDisplay(
    image_list=[
        sitk.LabelOverlay(img_T1_255, seg_implicit_thresholds),
        sitk.LabelOverlay(img_T1_255, seg_implicit_thresholds_clean),
    ],
    shared_slider=True,
    title_list=["before morphological closing", "after morphological closing"],
)
```

Quantification Continued

- Validating Registration results Visually

Intensity Based

```
print(f"Final metric value: {registration_method.GetMetricValue()}")
print(f"Optimizer's stopping condition, {registration_method.GetOptimizerStopConditionDescription()}")
```

```
Final metric value: -0.49414277378393795
Optimizer's stopping condition, GradientDescentOptimizerv4Template: Convergence checker passed at itera
```

Demon

```
Final metric value: 211830.4788376108
Optimizer's stopping condition, GradientDescentOptimizerv4Template: Convergence c
```

Quantification Continued

- Intensity Based Result:

Demon Result:

Quantification Continued

- Registration Evaluation:

The accuracy of nonrigid image registrations is commonly approximated using surrogate measures such as tissue label overlap scores, image similarity, image difference, or transformation inverse consistency error.

Quantifying the accuracy of nonrigid image registration is inherently difficult. Whereas a satisfactory gold standard database for evaluating the accuracy of rigid inter-modality registration of head images has been available for over a decade, no such gold standard yet exists for nonrigid registration.

Rigid registration accuracy is comparatively easy to quantify because the rigid registration error at any given point is completely determined by errors at three non-collinear landmarks.

Quantification Continued

- Registration Evaluation:

```
# Read label images
gs_label_image = sitk.ReadImage("./covid19.mha", sitk.sitkFloat32)
registered_label_image = sitk.ReadImage("./covid19_label.mha", sitk.sitkFloat32)
```

```
print(gs_label_image.GetSize())
print(registered_label_image.GetSize())
```

```
(512, 512, 412)
(512, 512, 70)
```

Calculate the Overlap

```
gs_label_arr = sitk.GetArrayFromImage(gs_label_image)
reg_label_arr = sitk.GetArrayFromImage(registered_label_image)

gs_area = sum(sum(sum(gs_label_arr)))
print(f'Gold Standard Area = {gs_area}')

overlap = 0
for z in range(registered_label_image.GetSize()[2]):
    for y in range(registered_label_image.GetSize()[1]):
        for x in range(registered_label_image.GetSize()[0]):
            if gs_label_arr[z,y,x] and reg_label_arr[z,y,x]:
                overlap += 1
print(f'Overlap = {overlap}')

coverage = overlap / gs_area
print(f'Coverage = {round(coverage*100, 2)} % ({coverage})')
```

```
Gold Standard Area = -68279470628.0
Overlap = 6393758
Coverage = -0.01 % (-9.364100133163674e-05)
```

Quantification Continued

- Registration Evaluation:

Information Measures and Overlap

- MI behavior varies
- MI Behavior Depends of Field of View:
  - can have clear maximum away from alignment

- When mutual information is used to evaluate the alignment of two finite images: overlap still has an confound:
  - Both Marginal entropies, $H(F)$ and $H(W)$ vary

- Alignment can be driven by choosing overlap that has large $H(F)$ and $H(W)$.
  - e.g. overlap with equal area of background and foreground intensities

- Rather than look for difference in joint and marginal entropies, use their ratio (like correlation coefficient):

$$Y(F,W)=\frac{H(F)+H(W)}{H(F,W)}$$

to be Maximized

Quantification Continued

- Compare results:

Region Growing

Watershed

Quantification Continued

- Registration Compare

Intensity Based

Demon

Thank You For Watching